INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.: (3) 954 90 20

Rapports de Recherche

N° 286

THE PARTITION MODEL: A DEDUCTIVE DATA BASE MODEL

Nicolas SPYRATOS

Avril 1984

THE PARTITION MODEL : A DEDUCTIVE DATA BASE MODEL

Nicolas SPYRATOS
University of Paris-South (LRI)
91405 Orsay, France

ABSTRACT : We present a new data model based on the lattice of partitions of an underlying population of individuals. The domains of our model are partitions, and the values are partition blocks, called "facts". A fact is true if and only if it is non-empty. Given a set of "atomic" partitions, "composite" facts are formed by intersections of atomic facts. The model is deductive, in the sense that we can deduce new facts from a given collection of facts, called a "data base", and from information coming from the model or the application being modeled. Semantic information, such as functional dependencies and ISA relationships can be incorporated in the deductive process, in the form of simple equations involving partitions.

RESUME : Nous présentons un modèle de données fondé sur le treillis de partitions d'une population d'individus. Les domaines de notre modèle sont les partitions alors que, les valeurs de ces domaines sont les blocks de partitions - que nous appelons "faits". Un fait est vrai si et seulement si il est non-vide. A partir d'un ensemble de partitions "atomiques" nous formons des faits "composites" par intersection de faits atomiques. Le modèle est déductif au sens où il permet la déduction de nouveaux faits, à partir d'un ensemble donné de faits - que nous appelons "base de données" - et d'informations provenant du modèle ou de l'application modélisée. La sémantique, telle que dépendances fonctionnelles ou relations ISA, peut être incorporée au processus déductif sous forme d'équations de partitions.

1. INTRODUCTION

We begin with an informal presentation of the basic concepts of our model. Throughout the presentation, we consider an underlying set Π, the population, whose elements we call individuals.
Given a characteristic of the individuals we can partition the population Π into disjoint subsets. Each subset contains all individuals having the same characteristic.

In Figure 1, the asterisks denote the individuals of Π. If these individuals are humans, then SEX is a relevant characteristic. With respect to SEX we can partition Π into two disjoint subsets, "female" and "male", containing the female and male individuals of Π, respectively (see Figure 1). It is important to note that, with respect to SEX, the individuals in the set "female" are indistinguishable and so are the individuals in the set "male". In a sense, SEX lumps into one block all individuals of the set "female", and into another block all individuals of the set "male". More formally, SEX is the name of α partition of the population Π into two subsets. The names of these subsets are "female" and "male", and we can write SEX = {female,male}.

Following a similar argument we can form the partitions AGE and STATUS shown in Figure 1. Thus to summarize so far, "characteristic X" stands for "partition X", and "a has the same characteristic X as b" stands for "a is in the same block of partition X as b". With the above understanding we shall use, henceforth, the term "partition" instead of "characteristic".

Given a set of partitions, we can form new partitions in a very natural way. Suppose for example that we are given the partitions AGE,SEX, and STATUS, shown in Figure 1. From AGE and SEX we can form a new partition by putting together all individuals having the same AGE and the same SEX. It follows that each block of the new partition is the intersection of one block from AGE and one block from SEX. Thus, if we denote the new partition by AGE.SEX, we can write :

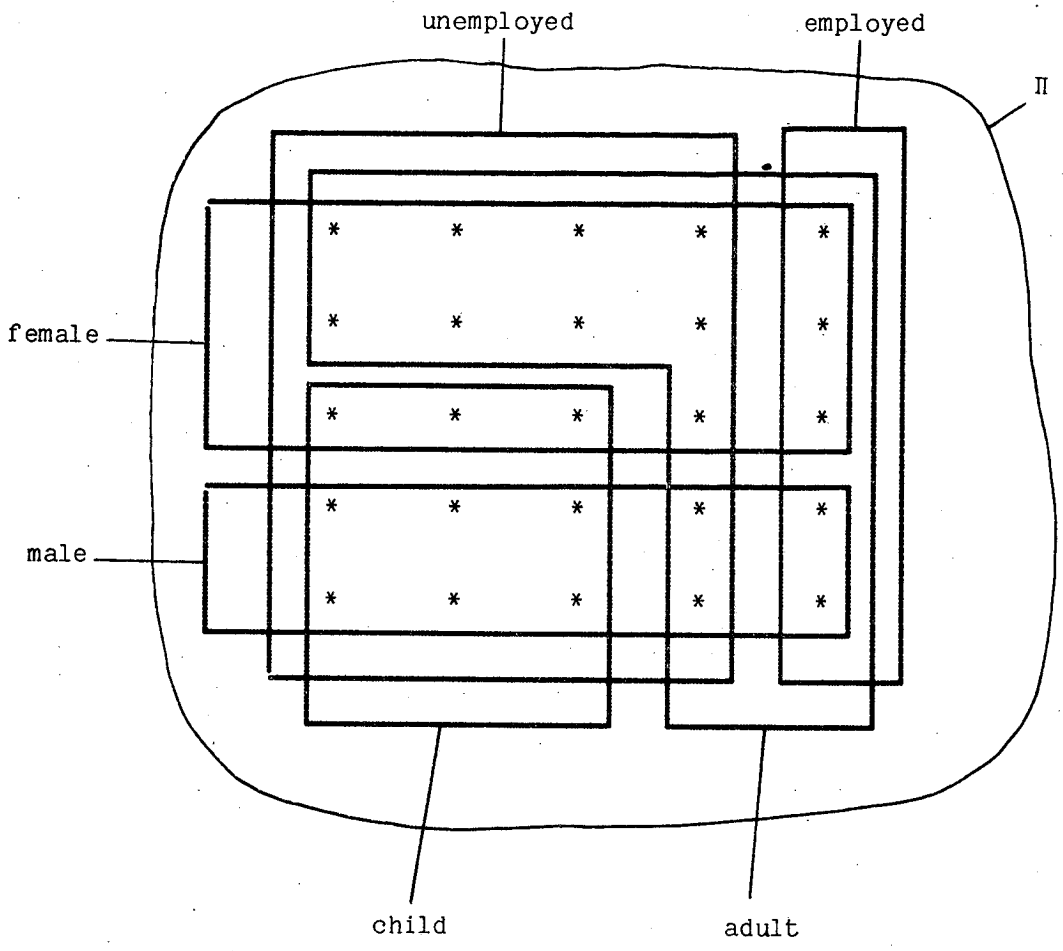
AGE.SEX = {child.female,child.male,adult.female,adult.male}

where, for each x in AGE and for each y in SEX, x.y denotes the intersection of x and y. Similarly, we can write :

AGE.STATUS={child.employed,child.unemployed,adult.employed,adult.unemployed}

The notation used for a new partition is quite suggestive, as it contains the "components" from which the new partition is formed. The same remark is true for the blocks of a new partition except that now we pay a price. There is a semantic pitfall, inherent in the notation for blocks. Indeed, the block "child.employed" of AGE.STATUS is empty (see Figure 1) and it should not appear as a block of AGE.STATUS. We shall come back to this remark repeatedly as it plays a fundamental role in the semantics of our model. By the way, we shall often use the terms "fact" instead of "block", "false" instead of "empty" and "true" instead of "non-empty". Thus, for example, the fact child.employed is false (see Figure 1).

We have just seen that "atomic" partitions, such as AGE, SEX, and STATUS, can serve as building blocks for "composite" partitions such as AGE.SEX, SEX.STATUS, AGE.STATUS, AGE.SEX.STATUS. Think now of an enterprise involving individuals of Π. In principle, all partitions listed above (both atomic and composite) are of interest to the enterprise during its operation. What kind of information should the enterprise maintain about a given partition ? To answer this question it is enough to observe that (in principle) the set of individuals present in the enterprise varies and so is, therefore, the set of blocks of any given partition. So the relevant information to maintain about any given partition is the set of its blocks present in the enterprise. Of course, any finite set of blocks may be present at a given moment. The set of blocks actually present is referred to as the current instance of the partition. Accordingly, the term instance of a partition means any (finite) subset of the partition.



AGE = {child,adult}

SEX = {female,male}

STATUS = {employed,unemployed}

Figure 1. Three partitions (i.e. characteristics) of the population Π

For example, the following set of blocks is an instance of the partition SEX.AGE :

{female.child,male.child,male.adult}

A more convenient way to represent an instance of a partition is to use a table. The heading of the table is the partition name and the lines are the blocks of the instance. Figure 2, shows the details of this representation. Thus, to summarize so far, the relevant information that the enterprise must maintain is the set of current instances of all partitions.

SEX.AGE
female.child
male.child
male.adult

or

SEX	AGE
female	child
male	child
male	adult

Figure 2. Tabular representation of the instance {female.child,male.child,male.adult} of the partition SEX.AGE.

Maintenance of information on the current instances of partitions requires collection of data and recording of facts (for example, in a computer). Figure 3 shows the recorded facts of two partitions, namely AGE.SEX and SEX.STATUS. We refer to such a collection of recorded facts as a data base. Of course, it is conceivable that while collecting data and recording facts some true facts are left out of the data base whereas, some false facts are recorded in the data base. Thus, roughly speaking, the data base can be seen as partial information on the current instances of the partitions. We can use the data base to try to deduce the current instances of the partitions. The process of deducing the current instance of a partition X from the data base, is referred to as quering the data base on partition X. The current instance of X is

referred to as the answer to the query. Let us query the data base of Figure 3, on the partition AGE.SEX.STATUS. Two steps are essential in answering the query

AGE	SEX
child	female
adult	male

SEX	STATUS
female	employed
male	employed

Figure 3. A data base

(a) Forming all possible facts of AGE.SEX.STATUS, using those available in the data base

(b) Deducing the true facts of AGE.SEX.STATUS, using every piece of available information.

The first step, requires operations for creating new facts from old. As such we use the traditional set theoretic operations. Thus, in our example, we can form all possible facts of AGE.SEX.STATUS, from those in the data base, using intersection. We obtain the following four facts :

1 : (child.female).(female.employed)
2 : (child.female).(male.employed)
3 : (adult.male).(female.employed)
4 : (adult.male).(male.employed)

To carry out the second step, deduction of true facts, we need additional information. This information is usually external to the data base, in the sense that it is not recorded in the data base. Such information can be model-oriented or application-oriented. An example of model-oriented information, inherent in our model, is the following :

"blocks of the same partition are disjoint"

Using this information we conclude that fact number 2, above, is false, and so is fact number 3. An example of application-oriented information is the following :

"the enterprise does not employee children"

Using this information we conclude that fact number 1 is false. Thus, we are left with fact number 4, that we can rewrite as follows :

4 : adult.male.employed

Another example of application-oriented information is the following assumption, on the conditions under which data is collected and facts are recorded

"if a fact is recorded in the data base then this fact is true"

Using this information, we conclude that fact number 4 is true. Thus, the answer to the query consists of fact number 4. It is important to note that, without the above assumption, the only thing that we can say is that the answer is contained between the empty set and the set consisting of fact number 4. In general, the answer to a query depends on the facts recorded in the data base and the external information. This concludes the informal presentation of the basic ingredients of our model.

A formal presentation of the model is given in Section 2. In Section 3, we introduce the basic operations of the model, namely product and sum. These operations allow the formation of new partitions from old, using intersection and union. In Section 4, we discuss the deductive mechanism of the model. We show that important semantic concepts, such as functional dependencies and ISA relationships, can be incorporated in the deductive mechanism in the form of simple equations. We introduce two assumptions on data, called "True Data" and "Complete Atomic Data", that lead to the

concept of "closure" of a partition instance. We then use closures to define the answer to a query. We show that the answer is not necessarily "complete". In the concluding Section 5 we discuss the prominent features of our model and suggest some further research.

It is important to note that our intention in this paper is to lay the foundations of a new data model and raise the relevant questions concerning its use. Specific problems related to various aspects of the model are dealt with in separate papers [5,6,7].

2. STRUCTURE OF THE MODEL

As we have seen in the introduction, partitions are the building blocks of our model. Let us then begin by recalling some definitions concerning partitions. Throughout the paper we consider an underlying set Π, the population, whose elements we call individuals. A partition of Π is a collection of non-empty disjoint subsets of Π whose union is Π. Henceforth, the term "partition" stands for "partition of Π, unless otherwise specified. We denote partitions by capital letters of the beginning of the alphabet, such as A, B,C, etc. We refer to members of a partition A as blocks or facts and we denote them by small letters, such as a, a_1, a_2, etc. If A and B are two partitions then the product of A and B, denoted A.B, is a partition defined as follows :

$$A.B = \{a.b \mid a \in A \text{ and } b \in B \text{ and } a.b \neq \emptyset\}$$

where a.b denotes the intersection of a and b. When no confusion is possible we write "AB" instead of "A.B", and "ab" instead of "a.b". It is easy to verify that the product is associative and commutative. As a consequence parentheses may be omitted and partition symbols may be permuted. For example, each of the symbols ABC and BCA denotes the product of A, B, and C.

Our goal in this section is to give a precise meaning to the tetm "data base", using the concept of a partition. To this end we need some additional definitions. First, suppose that an enterprise involving individuals of Π is interested in a specific partition

(i.e. characteristic). As the set of individuals involved in the enterprise changes with time, so does also the set of partition blocks that are present in the enterprise (a blocks is "present" if at least one of its individuals is involved in the enterprise). In principle, the only thing that we can assume about the set of blocks present in the enterprise is that it is finite. We refer to subsets of partitions as instances, and we denote them by small letters from the end of the alphabet, such as p, q, r, etc. The relevant information about a partition of interest to the enterprise is clearly the "current" instance of that partition. Now, the enterprise is usually interested in more than one partition. Usually, there are some "atomic" partitions (i.e. characteristics) of interest to the enterprise and some "composite" partitions, derived from the atomic partitions using products. For example, if A,B, and C are the atomic partitions then any of the following partitions could be of interest to the enterprise : A, B, C, AB, AC, ABC.

Definition 1. A universe U is a finite set of partitions, called atomic partitions. A partition on U is an atomic partition or the product of atomic partitions. A data base on U is a set of non-empty instances of partitions on U. □

In Figure 3, we give an example of a data base on the universe U = {A,B,C,D}. Partition instances are represented by means of tables (see also Figure 2). Usually, we do not specify the universe, as it is understood from the context. It is the set of all (atomic) partition symbols appearing in the data base.

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_1

B	C	D
b_1	c_1	d_2
b_2	c_1	d_1

A	C	D
a_1	c_1	d_2
a_2	c_1	d_2

Figure 3. A data base on the universe U = {A,B,C,D}

The reader familiar with the relational model will notice that the data base of Figure 3 looks exactly like a relational data base. This was done intentionally, through a careful choice of symbols and terms, in order to convince the reader that the structure of our model subsumes that of the relational model. Here are some correspondences of terms, between the two models that should help gain more insight :

attribute A	<—>	partition A (atomic)
domain of A	<—>	the set of blocks of A
value of A	<—>	block of A
relation schema R	<—>	partition R (composite)
tuple on R	<—>	block of R
relation on R	<—>	instance of R

We shall refer to the set of partitions appearing in a data base as the data base schema. Thus, in Figure 3, the data base schema is the set {ABC,BCD,ACD}. Note that, according to Definition 1, the data base schema may vary during the life of the enterprise being modeled.

We would like to stress here the fact that the similarity between the relational model and our model is only structural, that is they only appear the same. From the semantic point of view, the two models are completely different. Although the semantics of our model are discussed in detail later, we would like to give here a foretaste. First, in our model, a partition consists of a name, such as D in Figure 3, and of a set of blocks baring this name. For example, $D = \{d_1,d_2,d_3,d_4,d_5\}$. In the relational model, D is called a "domain" and additional names, called "attributes", are associated to D. These attributes play the role of variables taking their values in D. Second, in our model the d_i's are blocs of individuals whereas in the relational model the d_i's are indecomposable objects. This is true even in non-first-normal-form relations where the d_i's are indecomposable sets of objects. As a result, a tuple of the relational model is a set of indecomposable objects with unclear

semantics. To see this consider the tuples $a_1b_1c_1$ and $b_1c_1d_1$ of Figure 3. Relational theory always allows to join these tuples and produce a new tuple $a_1b_1c_1d_1$. If a_1, b_1, c_1, and d_1 are indecomposable, then the semantics of this new tuple is all the more unclear, as its very existence is doubtful ! Indeed, the existence of two individuals with characteristics a_1, b_1, c_1 and b_1, c_1, d_1 does not necessarily imply that of an individual with characteristics a_1, b_1, c_1, d_1. What is even worse is that we have no means of deciding the existence or non-existence of the new tuple $a_1b_1c_1d_1$. Now, in our model the tuples $a_1b_1c_1$ and $b_1c_1d_1$ are seen as intersections of blocks. Thus to say that an individual exists with characteristics a_1, b_1, and c_1 is tantamout to saying that $a_1b_1c_1 \neq \emptyset$. Assuming then that $a_1b_1c_1 \neq \emptyset$ and $b_1c_1d_1 \neq \emptyset$, we can deduce the existence of $a_1b_1c_1d_1$ from facts such as $b_1c_1 \subseteq d_1$. Formally,

$$(a_1b_1c_1 \neq \emptyset \text{ and } b_1c_1d_1 \neq \emptyset \text{ and } b_1c_1 \subseteq d_1) \Rightarrow a_1b_1c_1d_1 \neq \emptyset.$$

This kind of decutive reasoning is inherent in our model and constitutes one of its major advantages.

3. BASIC OPERATIONS

We introduce two operations, product and sum, that allow instances of partitions to be "multiplied" and "added". These operations produce new instances from old, using only set intersection and set union. Product and sum constitute the basic operations of our model.

Definition 2. Let q and r be instances of partitions A and B, respectively. The product of q and r, denoted q.r, is defined by

$$q.r = \{a.b \mid a \in q \text{ and } b \in r \text{ and } a.b \neq \emptyset\} \qquad \square$$

It is not difficult to see that the product is associative and commutative. When no confusion is possible we write "qr" instead of "q.r", and "ab" instead of "a.b". Figure 4 gives the "pictorial" product of two finite instances q and r. Note that the blocks of qr are "specializations" of those in q and r, as they carry more information about the individuals (individuals are denoted by

asterisks in the figure). Indeed, each block of qr is the intersection of two blocks, one from q and one from r. Thus, an individual in a block of qr belongs both to a block of q and to a block of r. And, supposing that q is an instance of A and r is an instance of B, we can say that the individual has both characteristics, A and B (recall that "characteristic" is a synonym for "partition"). This intuitive description motivates the following definition :

Definition 3. Let p, q, and r be the three partition instances. We say that p is a specialization of q and r if p=qr. □

When q=qr then q is a specialization of q and r that is, q is a specialization of r. In this case we usually say that q determines r. This term is justified as follows. If q=qr then for each block a of q and for each block b of r we have : $ab \neq \emptyset \Rightarrow a \subseteq b$ (immediate consequence of Definition 2). It follows that blocks of the form ab and ab' cannot co-exist in qr, as one of them must be empty. Thus, membership of an individual in the block a uniquely determines its membership in the block b.

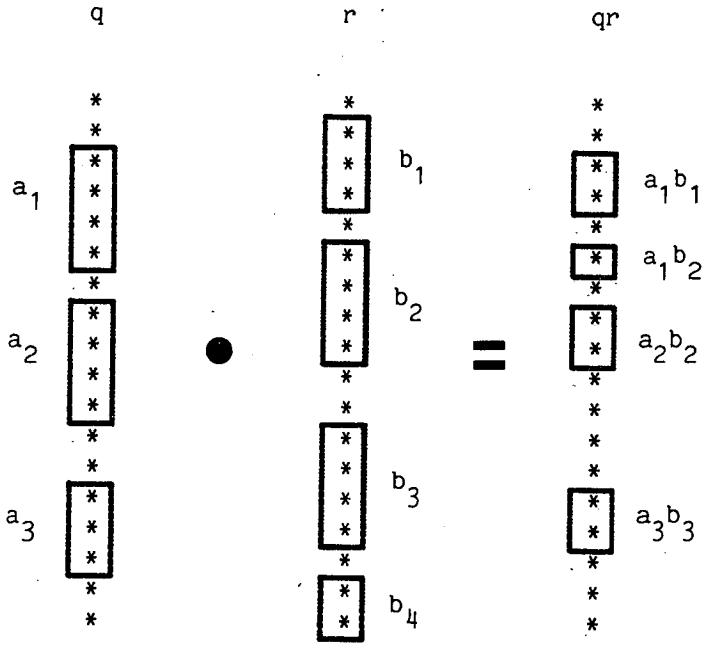


Figure 4. Pictorial multiplication of the instances $q = \{a_1, a_2, a_3\}$ and $r = \{b_1, b_2, b_3, b_4\}$.

It is instructive at this point to work out an example of product. We shall use the instances of ABC and BCD shown in Figure 3. Call these instances q and r, respectively. The first step in the computation of qr consists in writing down all possible intersections that could be in qr.

Step 1 : $qr \subseteq \{a_1b_1c_1b_1c_1d_2, a_1b_1c_1b_2c_1d_1, a_2b_1c_1b_1c_1d_2, a_2b_1c_1b_2c_1d_1, a_1b_2c_1b_1c_1d_2, a_1b_2c_1b_2c_1d_1\}$

From the "syntactic" point of view that is, finding all intersections that could be in qr, the computation is finished. From the "semantic" point of view, however, the computation has not yet started. Namely, we must decide now which of the candidate intersections are empty and which are not. And to do this we must look for relevant information, either in the model we are using or in the application that we are modelling. Here are three pieces of information that we shall use in our example :

I1 : blocks of the same partition are disjoint

I2 : if a fact (i.e. block) is recorded in the data base then this fact is true (i.e. non-empty)

I3 : ABC = BC

The first piece of information is model-oriented, in the sense that it is inherent in the structure of our model. The second is essentially an assumption on what is recorded in the data base, and as such it is application-oriented information. The third is also application-oriented information but it is less obvious why. In fact, it follows from I3 that each block of BC is contained in a block of A. Thus, for any individual in the enterprise, the characteristics (i.e. partitions) B and C "determine" the characteristic A. Of course, such information depends on the population of the specific enterprise, and as such it is application-oriented.

Going back to our example, let us see how we can use I1, I2, and I3, in order to carry out the "semantic" part in the computation

of qr. First, it follows from I1 that $b_1 b_2 = \emptyset$. Therefore, the set of candidates in Step 1 is now reduced as follows :

Step 2 : $qr \subseteq \{a_1b_1c_1d_2, a_2b_1c_1d_2, a_1b_2c_1d_1\}$

We have thus eliminated three intersections, from the original candidates for qr, because they were empty. What about the remaining three ? Let us consider the first of them, namely $a_1b_1c_1d_2$. It follows from I3 that

$$\forall\ bc \in BC\ \ \exists\ a \in A \text{ such that } bc \subseteq a$$

On the other hand, it follows fom I2 that $a_1b_1c_1 \neq \emptyset$. Combining these two facts we conclude that $b_1c_1 \subseteq a_1$, or equivalently that : $a_1b_1c_1 = b_1c_1$. Therefore, $a_1b_1c_1d_2 = b_1c_1d_2$ and, as $b_1c_1d_2 \neq \emptyset$ because of I2, we conclude that $a_1b_1c_1d_2 \neq \emptyset$. By similar arguments we can show that $a_2b_1c_1d_2 \neq \emptyset$ and $a_1b_2c_1d_1 \neq \emptyset$. Therefore, we can write

Step 3 : $qr = \{a_1b_1c_1d_2, a_2b_1c_1d_2, a_1b_2c_1d_1\}$

The reader familiar with the relational model will notice that the product just computed looks exactly like the relational join of q and r. This was done intentionally (through a careful choice of the example) in order to convince the reader that the product of our model subsumes the join of the relational model. In general, however, the situation is quite different. Suppose for example that instead of I3 we had the following piece of information

I3a : $a_1d_2 = \emptyset$ and $a_2d_2 = \emptyset$ and $a_1d_1 = \emptyset$

Then the product qr is empty ! An intermediate value of the product can be ibtained if I3 is replaced by the following piece of information

I3b : $a_1d_2 = \emptyset$ and $a_2d_2 \neq \emptyset$ and $a_1d_1 \neq \emptyset$

Then we find that : qr = $\{a_2b_1c_1d_2, a_1b_2c_1d_1\}$. Therefore, in the particular example that we are considering, the product qr varies between the following bounds :

$$\emptyset \subseteq qr \subseteq \{a_1b_1c_1d_2, a_2b_1c_1d_2, a_1b_2c_1d_1\}$$

The upper bound was obtained using only model-oriented information. Note that this upper bound is precisely the relational join of q and r. Two remarks are important at this point :

(1) The product qr can take any intermediate value between the two bounds, depending on the application-oriented information available to the computation

(2) The application-oriented information available to the computation may not be sufficient to actually compute the product. In most cases, however, such information reduces the upper bound. For example, suppose that I3 is replaced by

I3c : $a_1d_1 = \emptyset$

It follows from I3c that $a_1b_1c_1d_1 = \emptyset$, and this reduces the upper bound of qr as follows :

$$\emptyset \subseteq qr \subseteq \{a_2b_1c_1d_2, a_1b_1c_1d_2\}$$

Thus, althrough I1, I2, and I3c are not sufficient for the computation of qr, they reduce, nevertheless, the upper bound. On the other hand, note that adding to I3c the two equations : AB=A and CD=D will not result in a tighter upper bound.

Our discussion of products suggests the introduction of two "auxiliary" operations, projection and join, that simulate their synonyms of the relational model.

Definition 4. Let q and r be instances of partitions A and B, respectively. The projection of q over r, denoted $\pi_r(q)$, is defined by :

$$\pi_r(q) = r - \{b \in r \mid \{b\}.q = \emptyset\}$$

The join of q and r, denoted $q \bowtie r$, is defined by :

$$q \bowtie r = \{ab \mid a \in q,\ b \in r\} - \{ab \mid a \in q,\ b \in r,\ ab = \emptyset\} \quad \square$$

It is not difficult to see that : $q \bowtie r = \pi_q(r) \bowtie \pi_r(q)$. Figure 5 gives examples of projections and joins. Let us explain how we compute $\pi_r(q)$, in the first example. As $b_1 b_2 = \emptyset$, the block $b_2 c_1 d_1$ of r has empty intersection with each block of q. Therefore the block $b_2 c_1 d_1$ is excluded from $\pi_r(q)$. Nothing can be said about the remaining two blocks of r, using only model-oriented information. Indeed each of the remaining two blocks may have non-empty intersection with at least one block of q. Thus, both $b_1 c_1 d_1$ and $b_1 c_1 d_2$ are included in $\pi_r(q)$. In the second example, the computation of $\pi_r(q)$ is done using a similar argument. In the third example, where the projection of q is over the whole partition AC, the result coincides with relational projection of q over AC. In fact, it is not difficult to see that relational projection is a very special case of our projection. The last and final example of Figure 5 is the join of q and r. It is computed in exactly the same way as the relational join. Let us recall that the join of q and r is the best upper bound of qr that we can obtain, using only model-oriented information

A	B	C
a_1	b_1	c_2
a_2	b_1	c_1
a_1	b_1	c_1

q

B	C	D
b_1	c_1	d_1
b_1	c_1	d_2
b_2	c_1	d_1

r

B	C	D
b_1	c_1	d_1
b_1	c_1	d_2

$\pi_r(q)$

A	B	C
a_2	b_1	c_1
a_1	b_1	c_1

$\pi_q(r)$

A	C
a_1	c_2
a_2	c_1
a_1	c_1

$\pi_{AC}(q)$

A	B	C	D
a_2	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2

$q \bowtie r$

Figure 5. Projection and join in the partition model

To summarize, the relational join is a rigid upper bound of our product. More specifically, the join is the upper bound of the product when only model-oriented information is available. In a

sense, join "ignores" application-oriented information. We think that the inability of the join to incorporate application-oriented information is the source of the many semantic deficiencies of the relational model. However, we think that the join is the false culprit. In our opinion, the trouble lies at the very foundations of the relational model and it is related to the interpretation of domains and their values. This explains why repeated attempts to remedy semantic deficiencies, by considering the join as a culprit, have had little success (universal instance assumption and maximal objects are examples of such attmpts). The model proposed in this paper takes a radically different approach. Domains are interpreted as partitions of a population, values are interpreted as blocks of partitions and tuples are formed by intersection of values. The advantages of this approach are many

(1) The semantics of data and of operations on data are clear

(2) The duductive power of set theory becomes available to the model

(3) Application-oriented information can be incorporated in the decutive process in a straightforward manner.

We turn now to the second basic operation of our model, namely sum.

Definition 5. Let q and r be instances of partitions A and B, respectively. The sum of q and r, denoted q+r, is defined as follows : for each block x in $q \cup r$ do

(1) Let $c_1(x)=x$ and, for $i > 1$, let $c_{i+1}(x)=c_i(x) \cup (\cup \{z \in q \cup r \mid z.c_i(x) \neq \emptyset\})$

(2) Let $c(x)=c_i(x)$, for any i such that $c_{i+1}(x) = c_i(x)$

Then $q+r = \{c(x) \mid x \in q \cup r\}$. □

The computation of the sum is not really as difficult as it looks. Nevertheless, some remarks are necessary. First, let us observe that, for each block a in q, we have c(a)=a, unless $ab \neq \emptyset$ for some block b in r.That is, $\{a\}.r = \emptyset \Rightarrow c(a) = a$. Similarly, if b is in r then $q.\{b\} = \emptyset \Rightarrow c(b) = b$. It follows that, if the product qr is

finite then the computation of q+r terminates in a finite number of steps. And, if the product is empty then the sum is simply the union of the two instances that is ,

$$qr = \emptyset \Rightarrow q+r = q \cup r$$

Another remark is that, if $ab \neq \emptyset$ then $c(a) = c(b)$. Of course, when performing the sum of partition instances in a data base, we shall not worry for the infinite case. Figure 5 gives the "pictorial" sum of two finite instances q and r. Note that the blocks of q+r are "generalizations" of those in q and r, as they carry less information about the individuals (individuals are denoted by asterisks in the figure). Indeed, each block of q+r is the union of blocks from $q \cup r$. Thus, given an individual in a block of q+r the only thing that we can say is that the individual is either in a block of q or in a block of r (or both). If q is an instance of partition A (i.e. of characteristic A) and r is an instance of partition B (i.e. of characteristic B), then we can use a more

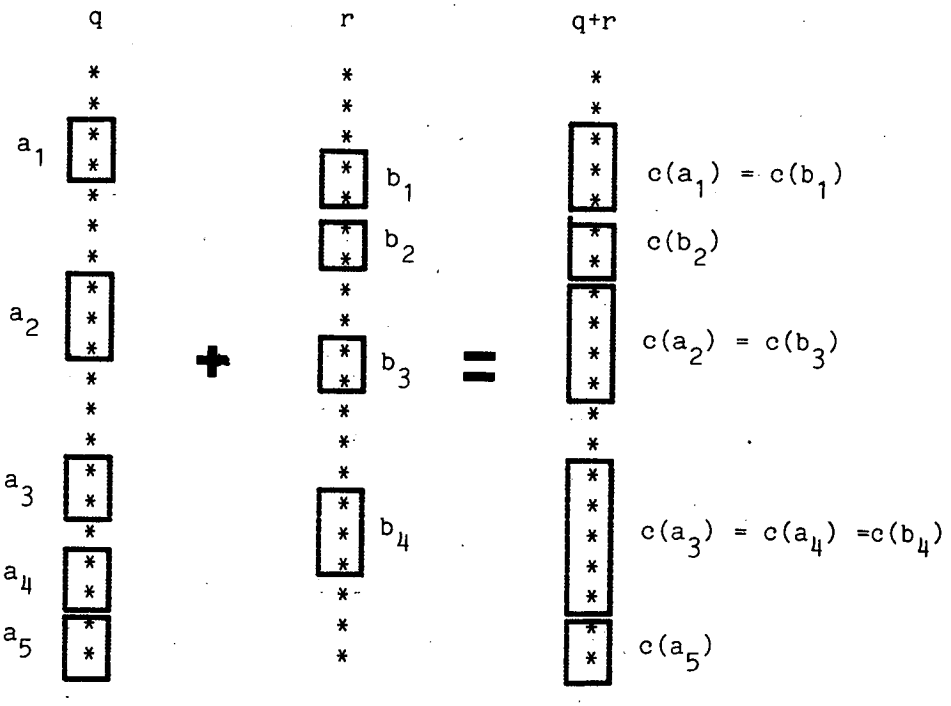


Figure 6. Pictorial addition of the instances $q=\{a_1,a_2,a_3,a_4,a_5\}$ and $r=\{b_1,b_2,b_3,b_4\}$.

descriptive language. Indeed, given an individual with characteristic A+B then that individual has either the characteristic A or the characteristic B (or both). This intuitive concept motivates the following definition.

Definition 6. Let p, q, and r be three partition instances. We say that p is a generalization of q and r if p=q+r. □

We hope that, in the light of our discussion so far, it is not difficult to see that the sum is an associative and commutative operation. Thus the "components" of a generalization can be added in any order, and parentheses can be omitted. Of course, if p is a generalization of q then q is a specialization of p.

Not surprisingly, all projections (in the relational sense) of a partition instance q are generalizations of q. Let us see an example. Refer to Figure 5 and let us compute the sum of q and $\pi_{AC}(q)$. We have :

$$q+\pi_{AC}(q)=\{a_1b_1c_2 \cup a_1c_2, a_2b_1c_1 \cup a_2c_1, a_1b_1c_1 \cup a_1c_1\}=\{a_1c_2, a_2c_1, a_1c_1\}=\pi_{AC}(q)$$

It follows from Definition 6 that $\pi_{AC}(q)$ is a generalization of q. In fact, we have the following theorem, which is an immediate consequence of the definitions of product, sum, and projection.

Theorem 1. Let q and r be instances of partitions. If qr=q then $q+\pi_r(q)=\pi_r(q)$. □

It is not difficult to see that if r is any projection of q (in the relational sense) then qr=q. Thus, it follows from Theorem 1 that all projections of q (in the relational sense) are generalizations of q. The converse of the above theorem is not true, as the following example shows. Let us compute the sum of two instances q and r defined as follows :

$$q = \{a_1b_1c_1, a_1b_2c_1\} \ , \quad r = \{b_1c_1d_1, b_2c_1d_1\}$$

In order to compute the sum we proceed as in the case of the product that is, we look for relevant information. Let us use the following three pieces of information

I1 : blocks of the same partition are disjoint
I2 : all facts contained in q and r are true
I3 : AD = A.

It follows from I3 that $a_1d_1 = a_1$, hence $(a_1b_1c_1)(b_1c_1d_1) = b_1c_1d_1$. It follows from I2 that $b_1c_1d_1 \neq \emptyset$, hence we must perform the union of $a_1b_1c_1$ and $b_1c_1d_1$:

$$a_1b_1c_1 \cup b_1c_1d_1 = b_1c_1(a_1 \cup d_1) = b_1c_1d_1$$

It follows from I1 that : $(b_1c_1d_1)(a_1b_2c_1) = \emptyset$. Therefore $b_1c_1d_1$ is in q+r. Similarly, we find that $b_2c_1d_1$ is in q+r and thus

$$q+r = \{b_1c_1d_1, b_2c_1d_1\} = r$$

Thus, r=q+r does not necessarily imply that r is a (relational) projection of q. To summarize, what Theorem 1 says is that all (relational) projections of an instance q are generalizations of q. On the other hand, what the preceding example says is that generalizations of q other than (relational) projections do exist and have a meaning.

We have seen so far that the two basic operations of our model, namely product and sum, subsume relational join and projection, respectively. The reader worrying about relational "selection" will be reassured by the following example. Let

$$q = \{a_1b_1c_1, a_2b_1c_2, a_1b_2c_2, a_1b_1c_2\}$$

Suppose first that we would like to select all those facts of q such that the fact b_1 is true. This is accomplished in our model by the product $q\{b_1\}$. Indeed, we have :

Selection on b_1 : $q\{b_1\} = \{a_1b_1c_1, a_2b_1c_2, a_1b_1c_2\}$

It is not difficult to see now how we can obtain selection on disjunction or conjunction of facts. Here are some examples :

Selection on a_1 or b_2 : $q\{a_1, b_2\} = \{a_1b_1c_1, a_2b_2c_1, a_1, b_2c_2, a_1b_1c_2\}$

Selection on a_1 and b_2 : $q\{a_1b_2\} = \{a_1b_2c_2\}$

We shall not pursue this discussion any further. We hope that the reader familiar with the relational model is by now convinced that the basic operations of our model, namely product and sum, subsume all relational oprations. Query languages based on product and sum are richer than relational languages because

(a) They can express all relational queries and

(b) They can express semantic information necessary for the computation of meaningful answers

The possibility of expressing semantic information using product and sum, is already implicit in definitions 3 and 5, and it will be further discussed in the next section. We believe that the inability of relational languages to express semantic information is due to the use of joins. As we have explained earlier, the join operation, by its very definition, "ignores" all application-oriented information. Now, such information is indispensable for the computation of meaningful answers, so it is "grafted" to the relational language by means external to the language (functional dependencies and maximal abjects are examples of such external means). The result is a rather ad hoc approach to some delicate semantic issues. But, once again, we believe that the join is not the real culprit. The trouble lies at the very foundations of the relational model, namely in the way domains and their elements are interpreted in that model.

We conclude this section by introducing an ordering that allows partition instances to be compared, with respect to their information content. In fact, this ordering has been implicit in our discussions so far.

Definition 7. Let q and r be instances of partitions A and B, respectively. We say that q is less than or equal to r, dnoted $q \leqq r$, if every block of q is contained in a block of r that is, if

$$\forall\, a \in q \quad \exists\, b \in r \text{ such that } a \subseteq b \qquad \square$$

It is not difficult to verify that the relation "$\leqq$" is a partial ordering on partition instances. The intuitive idea behind this ordering is the following. If the block a of q is a subset of the block b of r, then saying that an individual is in block a provides finer information than saying that the individual is in block b. In view of earlier discussions we would expect that if $q \leqq r$ then q is a specialization of r and r is a generalization of q. Indeed, this is confirmed by the following statement, which is an immediate consequence of the definitions of product and sum :

$$q \leqq r \quad \text{iff} \quad q = qr \quad \underline{\text{iff}} \quad r = q+r$$

With the two basic operations, product and sum, and the ordering just defined, the definition of our model is complete. Indeed, given a data base, we can now use

- the product, to produce specialisations of partition instances
- the sum, to produce generalizations of partition instances
- the ordering, to compare two partition instances.

For example, suppose that a data base is given on a universe U = {A,B,C}. Suppose also that the data base contains two instances, say q and r, of partitions AB and BC respectively. Then we can apply product and sum (on q and r) to compute instances for all partitions on U, namely for A, B, C, AB, BC, AC, ABC. Whether the computation

will produce exact values for those instances or simply bounds, depends on the information available to the deductive process. In the next section we discuss in more detail the deductive process and its main sources of information : the model, the application being modelled, and the data base.

In conclusion, the partial ordering just defined allows instances to be compared, whereas the product and sum allow instances to be "multiplied" and "added". These operations play a central role in our model and form a basic link with algebra and algebraic logic. This link between our data model and algebra is explored in a separate paper [7].

4. THE DEDUCTIVE MECHANISM

We have already seen, by way of examples, that the premises of deduction in our model are :

(a) the facts recorded in the data base

(b) information coming from the model or the application being modelled

What we deduce from those premises are instances of partitions (or bounds thereof). A partition whose current instance is of interest to a data base user is often referred to as a "query".

Definition 8. Given a data base on universe U, we call query any partition Q on U. We call answer to query Q, denoted $\alpha(Q)$, the current instance of Q. □

Figure 7 gives the complete picture of a hypothetical system based on our model. The term external information means information coming from the model and the application being modelled. In what follows, we discuss in more detail the four components of this system, namely "external information", "data base", "query" and "answer". We do not discuss the deductive mechansim. This mechanism is presumably a

machine (and, in the examples of our paper, this mechnism is just the author).

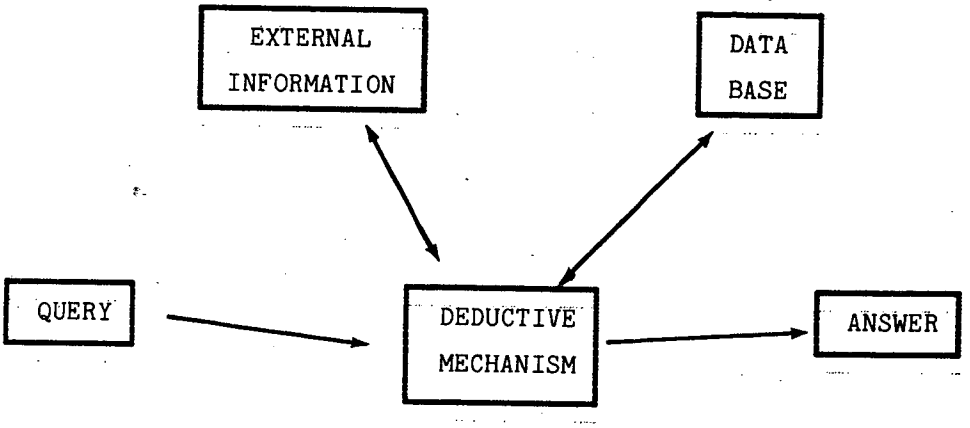


Figure 7. A query processing system based on our model

EXTERNAL INFORMATION

We have seen that in order to deduce new facts from those recorded in the data base we need information. In our discussions so far, we have identified two main sources of information, namely the model being used and the application being modelled.

Model-oriented information is information inherent in the structure of the model. In the examples so far we have used the following piece of information, inherent in the structure of our model :

I1 : blocks of the same partition are disjoint

It is not difficult to see that I1 can be expressed in the form of a simple equation as follows :

I1 : for any partition Q we have : $Q = Q.Q$

This kind of information does not depend on the application. That is, it does not depend on the specific enterprise being modelled but

but on the specific model being used. Once the model is chosen such information comes with the model and remains fixed.

Application-oriented information comes from the specific enterprise being modeled and as such it varies from one enterprise to another. Thus, whereas model-oriented information can be fixed, application-oriented information is variable by its very nature. We discuss here two sources of application-oriented information, namely the nature of data in the enterprise and enterprise practices.

Two important types of information, inherent in ata, have been identified and extensively studied in the literature. Informally, they correspond to the following statements :

an individual is female or male but not both
employees are persons
cars, boats, and planes are vehicles

The first statement is an example of "functional dependency", a concept extensively studied and used in relational data base theory (see for example [2]). The second and third statement are examples of "ISA relationship", a concept widely used in semantic modelling (see for example [4]). In our model, both types of information, functional dependency and ISA relationship, can be easily expressed in the form of simple equations using product and sum. In what follows we explain how.

Suppose that the underlying population consists of human individuals. Suppose also that NAME and SEX are two characteristics (i.e. partitions) of that population, defined as follows :

NAME = {John,Peter,Ann,Mary}, SEX = {Female,Male}

The statement "an individual is a female or male but not both" is easily interpreted in our model as follows : "a block of NAME has non-empty intersection with one and only one block of SEX". Equivalently, "each block of NAME is contained in a block of SEX".

And formally : NAME = NAME.SEX. In words, NAME is a specialization of SEX or, NAME determines SEX (see Definition 3). The term "functional dependency", used in the context of the relational model, can be easily justified as follows. As NAME = NAME.SEX, we can associate to each block b of NAME the unique block of SEX containing b. The resulting association is a function from the set NAME into the set SEX.

The second type of information mentioned earlier, namely ISA relationships, can also be expressed in our model in the form of simple equations. Speaking intuitively, the relationship "employees are persons" is an example of specialization because an individual is a person first, and secondarily an employee. To express this specialization formally let us define two partitions, EMPLOYEE and PERSON, as follows :

EMPLOYEE = NAME.SALARY, PERSON = NAME.ADDRESS

Here, NAME, SALARY, and ADDRESS are given partitions. An employee with name "John" and salary "20000" is represented by the block "John.20000", of the partition EMPLOYEE. If the employee's address is "Paris" then the block "John.Paris", of PERSON, holds this information. Now, if the employee is to inherit the characteristics of a person, then the block "John.20000" must be contained in the block "John.Paris". Indeed if this is the case, then we can write :

John.20000 = (John.20000).(John.Paris) = John.20000.Paris

In this way, the employee inherits the characteristics of a person. Now, if this is to be true for every employee, and not just for "John.20000", then we must require that : EMPLOYEE=EMPLOYEE.PERSON. Again this equation defines a function from the set EMPLOYEE into the set PERSON, exactly as in the case NAME=NAME.SEX, discussed earlier. There is, however, a subtle difference now. Namely, the population of which EMPLOYEE is a partition, can be a subset of the population of which PERSON is a partition. Of course, if the two populations coincide then there is no difference whatsoever between two cases.

The ISA relationship "cars, boats, and planes, are vehicles" is also easy to express in the form of a simple equation. Speaking intuitively, the basic individual populations are those of cars, boats, and planes, and they are put together to form the population of vehicles, for certain purposes. So, the relationship "cars, boats, and planes, are vehicles" is an example of a generalization. Formally, let CAR, BOAT and PLANE be three partitions of populations Π_c, Π_b, and Π_p, respectively. Then the partition VEHICLE, of the population $\Pi=\Pi_c \cup \Pi_b \cup \Pi_p$, defined by : VEHICLE = CAR+BOAT+PLANE, is a generalization of CAR, BOAT, and PLANE (see Definition 6).

To summarize, we have seen that two important types of information inherent in data, namely functional dependencies and ISA relationships, can easily be expressed in our model in the form of simple equations. This leads us to believe that relationships more complex than either functional dependencies or ISA relationships can also be expressed in our model, perhaps in the form of more complex equations.

We have seen so far application-oriented information coming from the nature of data. Enterprise practices constitute another source of application-oriented information. In both cases, the information can be expressed in the form of simple equations. There is, however, a difference. If the information comes from the nature of data then it is true for the whole population. As a result, the corresponding equation is an equation between partitions. On the other hand, if the information comes from enterprise practices then it is true only for the restricted population of the enterprise. As a result, the corresponding equation is an equation between partition instances. Consider, for example, the following pieces of information :

an employee has one and only one age
an employee has one and only one salary

The first, is information coming from the nature of data. As such it is true for the whole population. Formally, this information says

that the partition EMPLOYEE is a speicalization of the partition AGE that is, EMPLOYEE=EMPLOYEE.AGE. The second piece of information comes from enterprise practices. It is true for the restricted population of the enterprise. It may not be true for the population of a different enterprise. Formally, it says that the partition EMPLOYEE, restricted to the population of the enterprise, is a specialization of the partition SALARY, restricted to the population of the enterprise. These restrictions, are simply the current instances of EMPLOYEE and SALARY and, if we denote them by q and r respectively, then we can write q=qr. This equation express the information "an employee has one and only one salary". We will see examples of such equations later in this section.

We conclude our discussion on external information by presenting some simple properties of specializations. These properties are listed on the left of Table 1. On the right of the same table we have listed, for comparison purposes, the corresponding properties of functional dependencies - and their names - from the relational model. For the partition model, the symbols X, Y, Z, W, and V, must be interpreted as partitions, not necessarily atomic. For the relational model, the same symbols must be interpreted as sets of "attribute names", and juxtaposition of symbols, such as YZ, must be interpreted as set union. The properties of specializations listed in Table 1 are not difficult to prove. Here is the proof for projectivity :

$$X = XYZ \Rightarrow \forall\, x \in X \;\; x=xyz \Rightarrow \forall\, x \in X \;\; x=xy \Rightarrow X=XY \quad \text{Q.E.D.}$$

Now, using projectivity we can show pseudotransitivity as follows :

$$XZ = (XY)Z = X(YZ) = X(YZW) = XYZW = XZW \quad \text{Q.E.D.}$$

Using similar arguments, we can prove the remaining properties of Table 1. Questions concerning the axiomatization of specializations, and other related concepts, can also be asked in the context of the partition model. However, we shall not discuss these questions here.

Our intention in this paper is to present the foundations of the partition model. Specific questions related to various aspects of the model are discussed elsewhere [5,6].

PARTITION MODEL		RELATIONAL MODEL
$X = XX$	Reflexivity	$X \rightarrow X$
$X = XY \Rightarrow XZ = XZY$	Augmentation	$X \rightarrow Y \Rightarrow XZ \rightarrow Y$
$\left.\begin{matrix} X = XY \\ X = XZ \end{matrix}\right\} \Rightarrow X = XYZ$	Additivity	$\left.\begin{matrix} X \rightarrow Y \\ X \rightarrow Z \end{matrix}\right\} \Rightarrow X \rightarrow YZ$
$X = XYZ \Rightarrow X = XY$	Projectivity	$X \rightarrow YZ \Rightarrow X \rightarrow Y$
$\left.\begin{matrix} X = XY \\ Y = YZ \end{matrix}\right\} \Rightarrow X = XZ$	Reflexivity	$\left.\begin{matrix} X \rightarrow Y \\ Y \rightarrow Z \end{matrix}\right\} \Rightarrow X \rightarrow Z$
$\left.\begin{matrix} X = XY \\ YZ = YZW \end{matrix}\right\} \Rightarrow XZ = XZW$	Pseudo-transitivity	$\left.\begin{matrix} X \rightarrow Y \\ YZ \rightarrow W \end{matrix}\right\} \Rightarrow XZ \rightarrow W$
$\left.\begin{matrix} X = XYZ \\ Z = ZVW \end{matrix}\right\} \Rightarrow X = XYZV$	Accumulation	$\left.\begin{matrix} X \rightarrow YZ \\ Z \rightarrow VW \end{matrix}\right\} \Rightarrow X \rightarrow YZV$

Table 1. Some simple properties of specializations and their counterparts from the relational model

DATA BASE

The second component of the system in Figure 6 is the data base. A data base is a set of partition instances whose facts are used by the deductive mechanism. Whether these facts are true depends, to a large extent, on the person(s) collecting data in the enterprise and recording facts in the data base. Although data collection and recording is a function important to the enterprise, we shall not discuss it here. What is important, from our point of view, is that the deductive process needs some premises, and as such we assume the facts recorded in the data base. Hence the following assumption :

True Data (TD) : If a fact is recorded in the data base then this fact is true

From a mathematical viewpoint the facts, recorded in the data base are the axioms of our deductive process. In our examples, we shall always make the true-data assumption.

Of course, it is conceivable that some true facts of data base partitions are left out of the data base, for various reasons. We shall distinguish two cases :

Case 1 : True facts that are not in the data base but are implied by those in the data base and the external information

Case 2 : True facts that are not in the data base and are not implied by those in the data base and the external information.

Let us see an example. Consider the data base shown in Figure 8, with external information A=AB and B=BC (henceforth, we shall not refer explicitly to model-oriented information, although we shall be using it, of course). The block a_2b_2, of AB, is an example of Case 1. Indeed, a_2b_2 is not in the data base but it is implied by data base facts and the external information, as follows :

True Data => $a_2b_2c_1 \neq \emptyset$
($a_2b_2c_1 \neq \emptyset$ and B=BC) => $a_2b_2c_1 = a_2b_2$
Therefore, $a_2b_2 \neq \emptyset$ Q.E.D.

The question is : what are the candidates for Case 1 ? In Figure 8, for example, it would seem that each of the following facts is a

EXTERNAL INFORMATION

A = AB
B = BC

DATA BASE

	A	B	C
q	a_1	b_1	c_1
	a_2	b_2	c_1

	A	B
r	a_3	b_2

Figure 8. A data base example

candidate for Case 1 :

$$a_1b_2c_1, a_2b_1c_1, a_3b_1c_1, a_3b_2c_1, a_1b_1, a_1b_2, a_2b_1, a_2b_2, a_3b_1$$

In the other words, it would seem that the candidates for Case 1 are all those facts (of data base partitions) that one can form by intersections of atomic facts appearing in the data base. For example, the following facts appear in the data base of Figure 8 :

a_1, a_2, and a_3, of atomic partition A
b_1 and b_2, of atomic partition B
c_1, of atomic partition C

Now, we can combine these atomic facts, in all possible ways, to derive facts of partition ABC and of partition AB. The resulting sets of facts are shown in Figure 9, where they are denoted by $\bar{q}$ and $\bar{r}$. Thus, all facts in the sets $\bar{q}$-q and $\bar{r}$-r are candidates for Case 1.

	A	B	C	
q	a_1	b_1	c_1	$\bar{q}$
	a_2	b_2	c_1	
	a_1	b_2	c_1	
	a_2	b_1	c_1	
	a_3	b_1	c_1	
	a_3	b_2	c_1	

	A	B	
r	a_3	b_2	$\bar{r}$
	a_1	b_1	
	a_1	b_2	
	a_2	b_1	
	a_2	b_2	
	a_3	b_1	

Figure 9. The closures of the partition instances of Figure 8.

There is a systematic way for computing the sets $\bar{q}$ and $\bar{r}$. Here are the basic steps :

Step 1 : Collect all facts of atomic partitions using projections and unions. In the example of Figure 8 we have :

$$P_A = \pi_A(q) \cup \pi_A(r) = \{a_1, a_2\} \cup \{a_1, a_3\} = \{a_1, a_2, a_3\}$$

$$P_B = \pi_B(q) \cup \pi_B(r) = \{b_1, b_2\} \cup \{b_2\} = \{b_1, b_2\}$$

$$P_C = \pi_C(q) \cup \pi_C(r) = \{c_1\} \cup \emptyset = \{c_1\}$$

Step 2 : Take the join of the results in Step 2. Continuing with the example of Figure 8, we have :

$$p = p_A \bowtie p_B \bowtie p_C = \{a_1b_1c_1, a_1b_2c_1, a_2b_1c_1, a_2b_2c_1, a_3b_1c_1, a_3b_2c_1\}$$

Step 3 : Take projection of p over every data base partition. Continuing with the example of Figure 8, we have :

$$\bar{q} = \pi_{ABC}(p) = p$$

$$\bar{r} = \pi_{AB}(p) = \{a_1b_1, a_1b_2, a_2b_1, a_2b_2, a_3b_1, a_3b_2\}$$

The results obtained in Step 3 are those shown in Figure 9. Sets of facts, such as $\bar{q}$ and $\bar{r}$, play a central role in our model. Therefore, we shall dignify them with a name.

Definition 9. Let $U = \{A_1, A_2, \ldots, A_m\}$ be a universe. Let $\Delta = \{Q_1, Q_2, \ldots, Q_n\}$ be a schema on U. Let $\delta = \{q_1, q_2, \ldots, q_n\}$ be a data base on Δ. For i=1,2,...,m, let

$$p_i = \pi_{A_i}(q_1) \cup \pi_{A_i}(q_2) \cup \ldots \cup \pi_{A_i}(q_n)$$

The closure of δ, denoted $\bar{\delta}$, is defined by : $\bar{\delta} = p_1 \bowtie p_2 \bowtie \ldots \bowtie p_m$. For i=1,2,..,n, the closure of q_i, denoted $\bar{q}_i$, is defined by : $\bar{q}_i = \pi_{Q_i}(\bar{\delta})$. For i=1,2,...,m, q_i is closed if $q_i = \bar{q}_i$ □

It is important to note that the closure $\bar{\delta}$ is not essential in the definition of $\bar{q}_i$'s. It was simply introduced for convenience in the computation of $\bar{q}_i$'s. Indeed, the closure $\bar{q}_i$ can be defined, without defining $\bar{\delta}$, as follows. Let $Q_i = A_{i_1}A_{i_2}\ldots A_{i_k}$. Then $\bar{q}_i$ is defined by : $\bar{q}_i = p_{i_1} \bowtie p_{i_2} \bowtie \ldots \bowtie p_{i_k}$. Thus, the reader familiar with the relational model is warned against any confusion between Definition 9 and universal instance assumptions of various kinds.

To summarize so far, the facts recorded in the data base are simply some true facts among those currently true. The remaining true facts will have to be deduced from those in the data base and the external information. The question is what facts, not recorded in the data base, might possibly be true. We have seen that facts not recorded in the data base but belonging to closures of data base instances can be true. Then the crucial question is the following. Are there facts (of data base partitions) not belonging to the closures of data base instances that might possibly be true ? The answer is no, if we assume that the atomic facts appearing in the data base are the only atomic facts currently true. Hence, the following assumption concerning atomic facts :

Complete Atomic Data (CAD) : If an atomic fact is true then this fact is recorded in the data base.

Let us suppose CAD in the example of Figure 8. Given a fact not in $\bar{q}$ or $\bar{r}$, say the fact $a_4b_1c_1$, we can prove it false as follows :

CAD => $a_4 = \emptyset$
Therefore, $a_4b_1c_1 = \emptyset$ Q.E.D.

This proof can be generalized. Indeed, let f be a fact (of data base partitions) not belonging to the closures of data base instances. It follows that at least one atomic fact, say a, of f does not appear in the data base. Assuming CAD, it follows that the atomic fact a is false (i.e. a=Ø). As f is an intersection of atomic facts, it follows that f is false (i.e. f=Ø).

It is important to have a clear understanding of the CAD assumption, as we shall always adopt it from now on. So, let us discuss it in more detail. In the data base of Figure 8, the TD assumption (recall that TD stands for "True Data") implies that the following facts are true :

$a_1, a_2, a_3, b_1, b_2, c_1, a_1b_1, a_2b_2, a_3b_2, b_1c_1, b_2c_1, a_1c_1, a_2c_1, a_1b_1c_1, a_2b_2c_1$

On the other hand, the CAD assumption implies that, among all facts of the atomic partitions A, B, and C, the only ones that are currently true are : $a_1, a_2, a_3, b_1, b_2, c_1$. It says nothing about non-atomic facts. If we restrict our attention to atomic facts, then the combined TD and CAD assumption is equivalent to the following statement :

"an atomic fact is true iff this fact is recorded in the data base"

Speaking roughly, what TD and CAD says is that :

(a) all facts recorded in the data base are true
(b) although some true non-atomic facts may have been left out of the data base, all true atomic facts are recorded in the data base

In relational database theory two assumptions, known as the "Closed World Assumption" and the "Open World Assumption", have been widely used (see for example [3]). According to the closed world assumption, every fact recorded in the data base is true while, every fact not recorded in the data base is false. Therefore, this assumption corresponds to the following statement :

"a fact is true iff this fact is recorded in the data base"

Now, referring to Figure 8, the closed world assumption implies that the fact $a_3b_2c_1$, of ABC, is false (as it is not recorded in the instance of ABC). Yet, this fact can be shown to be true, using the data base, the external informations and ... the closed world assumption !! Here is the proof :

Closed World Assumption => $a_2b_2c_1 \neq \emptyset$ and $a_3b_2 \neq \emptyset$
($a_2b_2c_1 \neq \emptyset$ and B=BC) => $b_2 \subseteq c_1$
($b_2 \subseteq c_1$ and $a_3b_2 \neq \emptyset$) => $a_3b_2c_1 \neq \emptyset$ Q.E.D.

This is an amazing result. It shows that, in the presence of external information, the closed world assumption is not just unrealistic, it is simply wrong ! It is important to note the ease by which the deductive mechanism of our model led to this conclusion. On the other hand, the lack of deductive mechanism in the relational model makes it almost impossible to discover such basic contradictions. There is one case in which it is safe to make the closed world assumption. Namely, when every data base instance q is closed (recall that q is closed if $q=\bar{q}$). However, such cases almost never arise in practice for the following reason. Even if an instance is closed, this property is destroyed (in general) by the insertion or deletion of facts. On the other hand, as we shall see later, it is not always possible to compute the closure of an instance (and in cases where this is possible, the relational model has no deductive mechanism for computing the closure).

The second major assumption used in relational data base theory is the open world assumption. According to this assumption, every fact recorded in the data base is true while, nothing is assumed about facts not recorded in the data base. Therefore, the open world assumption corresponds to the following statement :

"if a fact is recorded in the data base then this fact is true"

Note that this is precisely our true-data assumption. The open world assumption does not lead to contradictions, as does the closed world assumption. Nevertheless, it is a very rough assumption, as it treats in the same way <u>all</u> facts that are not recorded in the data base. We have seen earlier that a fact not recorded in the data base falls into one of two distinct cases, that we have called Case 1 and Case 2. The concept of closure helps formalize this distinction. Indeed, if q is a data base instance of a partition Q, then a fact of Q not in q (i.e. not recorded in the data base) is either in $\bar{q}-q$ or outside $\bar{q}$. Clearly, if a fact f is in $\bar{q}-q$ then there is hope to prove f true or false, depending on the facts in q and the external information available. On the other hand, if f is outside $\bar{q}$ then there is no hope whatsoever of proving f true or false, and we must <u>assume</u> f true or f false. The open world assumption does not

make this distinction and this is why we think that it is a very rough assumption.

We believe that the right frontiers, on which we can reasonably make assumptions, are not those defined by the data base instances but, rather, those defined by the closures of data base instances. Using closures, we can give now a formal statement of the assumptions that we consider reasonable.

Definition 10. Let $U = \{A_1, A_2, \ldots, A_m\}$ be a universe. Let $\Delta = \{Q_1, Q_2, \ldots, Q_n\}$ be a schema on U. Let $\delta = \{q_1, q_2, \ldots, q_n\}$ be a data base on Δ. For i=1,2,...,m let

$$p_i = \pi_{A_i}(q_1) \cup \pi_{A_i}(q_2) \cup \ldots \cup \pi_{A_i}(q_m)$$

True Data (TD) : $\forall f \in Q_i \quad f \in q_i \Rightarrow f \neq \emptyset, \quad i=1,2,\ldots,n$

Complete Atomic Data (CAD) : $\forall f \in A_i \;, f \notin p_i \Rightarrow f=\emptyset, \; i=1,2,\ldots,m$ □

It is not difficult to see that admitting TD and CAD implies the following statement :

$$\forall f \in A_i \quad (f \in p_i \Leftrightarrow f \neq \emptyset), \quad i=1,2,\ldots,m$$

A remark concerning definitions 9 and 10 is in order at this point. In our examples so far we have used external information only in the form of equations involving partitions and not involving specific facts. For example, referring to Figure 8, consider adding the following piece of information : $a_1c_1 \neq \emptyset \Rightarrow b_5 \neq \emptyset$. The question is what to do with b_5. The answer is that all facts known to be true, such as b_5 in this example, must be recorded in the data base or, at least, must be taken into account when applying definitions 9 and 10.

The final issue concerning the data base as a premise of deduction, is that of consistency. The data base is only one of the premises of deduction in our model. The other premise of deduction

is external information. We must ensure that there are no contradictions between these two premises. We call a data base consistent if there are no contradictions between facts recorded in the data base and external information. Of course, whenever a (consistent) data base is updated that is, whenever a fact is inserted, deleted, or modified, consistency must be verified. Consistency checking and data base updating are discussed in a separate paper [6].

QUERY PROCESSING

A query in our model is a partition Q whose current instance is of interest to a data base user. The current instance of Q, denoted $\alpha(Q)$, is the answer to the query. The premises from which we compute the answer $\alpha(Q)$ are the data base and the external information. The means by which we compute $\alpha(Q)$ are the basic operations of set theory, namely intersection, union and complement, and the two operations defined earlier, namely product and sum. Speaking roughly, product and sum are simply intersection and union extended to some special kind of set families called partitions. Thus, the operations that we are using are, essentially, the basic operations of set theory. When computing answers to queries we assume true data (TD), complete atomic data (CAD), and a consistent data base.

First, assume that the query Q is a data base partition. Then we can express its answer in terms of closures as follows. As the answer $\alpha(Q)$ is just the current instance of Q, we can write

$$\alpha(Q) = \{f \in Q \mid f \neq \emptyset\}$$

On the other hand, let q be the set of facts of Q recorded in the data base. Then the assumptions of true data and complete atomic data imply that : if f is not in $\bar{q}$ then f is false. Hence, we can write :

$$a(Q) = \{f \in \bar{q} \mid f \neq \emptyset\}$$

Now, the facts of q are known to be true and, therefore, we can write :

$$\alpha(Q) = q \cup \{f \in (\bar{q}-q) \mid f \neq \emptyset\}$$

Some (or even all) facts of $\bar{q}-q$ might be true. We have already seen an example of this kind, in the data base of Figure 8. Indeed we have proved that the fact $a_3b_2c_1$ is true. This fact belongs to $\bar{q}-q$ (see also Figure 9) and, as it is true, it belongs to the answer $\alpha(ABC)$. Now, the question is : can we always decide which facts of $\bar{q}-q$ are true and which are false ? The answer to this question depends on the data base, the external information, and the query being processed. Let us see some examples. Consider the data base of Figure 10, and let us compute the answer to query Q=ABC. To do this we must consider the facts of $\bar{q}-q$, one by one, and decide whether they are true or false. The fact $a_1b_2c_1$, of $\bar{q}-q$, is false because it contradicts q and A=AB. Here is the proof :

True Data $\Rightarrow a_1b_1c_1 \neq \emptyset$
$(a_1b_1c_1 \neq \emptyset$ and A=AB$) \Rightarrow a_1 \subseteq b_1$
$a_1 \subseteq b_1 \Rightarrow a_1b_2 = \emptyset$
$a_1b_2 = \emptyset \Rightarrow a_1b_2c_1 = \emptyset$ Q.E.D.

By similar arguments we can show that the facts $a_2b_1c_1$ and $a_3b_1c_1$ are false. However, the remaining fact of $\bar{q}-q$, namely $a_3b_2c_1$, can be shown true as follows :

True Data $\Rightarrow a_3b_2 \neq \emptyset$ and $b_2c_1 \neq \emptyset$ and $a_3 \neq \emptyset$
$(a_3b_2 \neq \emptyset$ and A=AB$) \Rightarrow a_3 \subseteq b_2$
$(b_2c_1 \neq \emptyset$ and B=BC$) \Rightarrow b_2 \subseteq c_1$
$(a_3 \neq \emptyset$ and $a_3 \subseteq b_2$ and $b_2 \subseteq c_1) \Rightarrow a_3b_2c_1 = a_3 \neq \emptyset$ Q.E.D.

EXTERNAL INFORMATION

A=AB
B=BC

DATA BASE

	A	B	C
q	a_1	b_1	c_1
	a_2	b_2	c_1

	A	B	D
r	a_1	b_1	d_1
	a_3	b_2	d_1

CLOSURES

	A	B	C
q	a_1	b_1	c_1
	a_2	b_2	c_1
	a_1	b_2	c_1
	a_2	b_1	c_1
	a_3	b_1	c_1
	a_3	b_2	c_1

$\bar{q}$

	A	B	D
r	a_1	b_1	d_1
	a_3	b_2	d_1
	a_1	b_2	d_1
	a_2	b_1	d_1
	a_2	b_2	d_1
	a_3	b_1	d_1

$\bar{r}$

Figure 10. An example data base

Thus, we have been able to decide, for every fact of $\bar{q}$-q, whether it is true or false. Therefore the answer to query Q=ABC is the following :

$$\alpha(ABC) = \{a_1b_1c_1, a_2b_2c_1, a_3b_2c_1\}$$

It is important to note that, although the query ABC is a data base partition, the answer is not the data base instance q, as one might expect. It is also important to note that, if the data base of Figure 10 is viewed as a relational data base, there is no way of concluding that the fact $a_3b_2c_1$ is in $\alpha(ABC)$, using relational operators.

Let us now compute the answer to query Q=ABD. Proceeding as in the case of ABC, it is not difficult to show tha the facts $a_1b_2d_1$, $a_2b_1d_1$, and $a_3b_1d_1$, of $\bar{r}$-r, are false. Here are some hints for the proofs :

$a_1b_2d_1$ contradicts r and A=AB
$a_2b_1d_1$ contradicts q and A=AB
$a_3b_1d_1$ contradicts r and A=AB

However, the remaining fact of $\bar{r}$-r, namely $a_2b_2d_1$, cannot be shown false nor true. Therefore, we cannot give a "complete" answer to query ABD. We can only give bounds of $\alpha(ABD)$, as follows :

$$\{a_1b_1d_1, a_3b_2d_1\} \subseteq \alpha(ABD) \subseteq \{a_1b_1d_1, a_3b_2d_1, a_2b_2d_1\}$$

Now, suppose that the information BD = BDA becomes available, in addition to A=AB and B=BC. Then the fact $a_2b_2d_1$ can be shown to be false (it contradicts r and BD=BDA). In this case, the answer $\alpha(ABD)$ is complete. On the other hand, suppose that the information B=BD becomes available, instead of BD=BDA. Then the fact $a_2b_2d_1$ can be shown true as follows :

True Data $\Rightarrow$ $b_2d_1 \neq \emptyset$ and $a_2b_2 \neq \emptyset$
$(b_2d_1 \neq \emptyset$ and B=BD$) \Rightarrow b_2 \subseteq d_1$
$(a_2b_2 \neq \emptyset$ and $b_2 \subseteq d_1) \Rightarrow a_2b_2d_1 = a_2b_2 \neq \emptyset$ Q.E.D.

In this case, the answer $\alpha(ABD)$ is again complete, although it is a different answer than in the previous case. In general, the "completeness" of the answer depends on the data base and the external information.

Definition 11. Let U be a universe. Let $\Delta = \{Q_1, Q_2, \ldots, Q_n\}$ be a schema on U. Let $\delta = \{q_1, q_2, \ldots, q_n\}$ be a data base on Δ. Let ε be external information on U and δ. For every Q in Δ define :

$$\alpha_+(Q) = q \cup \{f \in (\bar{q}-q) \mid (\delta \text{ and } \varepsilon) \Rightarrow f \neq \emptyset\}$$
$$\alpha_-(Q) = \{f \in (\bar{q}-q) \mid (\delta \text{ and } \varepsilon) \Rightarrow f = \emptyset\}$$

A query Q of Δ is called complete, with respect to δ and ε, if $\bar{q} = \alpha_+(Q) \cup \alpha_-(Q)$. □

In general, the answer to a query Q lies between $\alpha_+(Q)$ and $\bar{q} - \alpha_-(Q)$, that is

$$\alpha_+(Q) \subseteq \alpha(Q) \subseteq \bar{q} - \alpha_-(Q)$$

We say that the data base is complete when all the queries in the data base schema are complete. The data base of Figure 11 is an example of a complete data base. In fact, the answers to AB and BC are the data base instances.

We have seen so far examples of queries that are data base partitions. When a query P is not a data base partition then the answer $\alpha(P)$ is computed from the answers of the data base partitions using product and sum. However, even if the data base is complete, the query P may not be complete. Let us see an example. Suppose that we want to compute the answer to ABC, in the data base of Figure 11.

EXTERNAL INFORMATION	DATA BASE		CLOSURES	
	A B	B C	A B	B C
C=CB	$q\begin{cases} a_1 & b_1 \\ a_1 & b_2 \end{cases}$	$r\begin{cases} b_1 & c_1 \\ b_1 & c_2 \end{cases}$	$\bar{q}\begin{cases} a_1 & b_1 \\ a_1 & b_2 \end{cases}$	$\bar{r}\begin{cases} b_1 & c_1 \\ b_1 & c_2 \\ b_2 & c_1 \\ b_2 & c_2 \end{cases}$

Figure 11. A complete data base.

As AB and BC are complete, we have :

$$\alpha(ABC) = \alpha(AB).\alpha(B) = q.r \qquad q \bowtie r = \{a_1b_1c_1, a_1b_1c_2\}$$

The external information C=CB is not enough to decide whether the facts $a_1b_1c_1$ and $a_1b_1c_2$ are true or false. Thus ABC is not complete, although the data base is complete. It would be interesting to know under what conditions complete queries, such as AB and BC, produce complete answers for queries, such as ABC. The following theorem gives a sufficient condition.

Theorem 2. Let XY and YZ be complete queries, where X, Y, and Z are not necessarily atomic partitions. If Y=YX or Y=YZ then

$$\alpha(XY).\alpha(YZ) = \alpha(XY) \bowtie \alpha(YZ) \qquad \square$$

The simple rule suggested by this theorem can be used to identify sets of data base partitions that behave "nicely". That is, if the partitions in such a set are complete then products of partitions in the set can be replaced by joins. Let us see an example. Figure 12 gives a pictorial representation of a data base schema Δ and external information ε. A partition of Δ is denoted by its atomic partitions in a circle. An equation of ε of the form X=XY, is denoted by $X \rightarrow Y$. Now, given a cmplete data base on Δ, we can apply Theorem 2, to conclude that :

$\alpha(AB).\alpha(BC) = \alpha(AB) \bowtie \alpha(BC)$ [because B=BA]

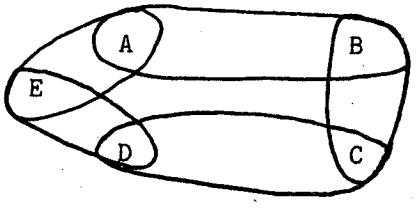
$[\alpha(AB).\alpha(BC)].\alpha(CD) = [\alpha(AB) \bowtie \alpha(BC)] \bowtie \alpha(CD)$ [because C=CB=CBA]

$\alpha(AE).\alpha(ED) = \alpha(AE) \bowtie \alpha(ED)$ [because E=ED]

Thus, the schema Δ is partitioned into two disjoint subschemas, Δ_1 = {AB,BC,CD} and Δ_2 = {AE,DE}. Note that neither Δ_1 nor Δ_2 can be further enlarged by a new application of Theorem 2. In a sense, they are "maximal". What Theorem 2 implies is that the results of joins within each subschema contain only true facts. What Theorem 2 does <u>not</u> imply is that a query can be answered using only one of the two subschemas. We shall not pursue this subject here. Our intention in this paper has been to present the foundations of the partition model and give evidence of its deductive power, by way of examples.

Δ = {AB,AE,BC,CD,DE}

ε : B=BA <u>and</u> C=CB <u>and</u> E=ED



<u>Figure 12.</u> In the subschemas Δ_1 = {AB,BC,CD} and Δ_1 = {AE,DE}, products can be replaced by joins.

Specific questions related to various aspects of the model are treated formally in separate articles [5,6].

5. CONCLUDING REMARKS

We have presented the basic aspects of a new data model, that we call the partition model. Its basic novelty is the interpretation of data domains and data values. Domains are partitions of an underlying population, and values are partition blocks, also called facts. Composite partitions are built from atomic partitions, and composite facts are built from atomic facts. The operations used are the basic operations of set theory. As we have explained, this process leads to a deductive data base model, in a simple and straighforward manner.

A data value (or fact, or block) of the partition model has both a name and a meaning. The name is just a label, or identifier. The meaning is that a value is a set of individuals, of an underlying population. If we ignore the meaning of values then what we are left with is the syntactic part of the partition model. And we have seen that the syntactic part of the partition model subsumes the relational model. By taking into account both name and meaning of values, the partition model offers significant advantages :

(1) The semantics of data and of operations on data are clear

(2) The deductive power of set theory becomes available to the model

(3) Application-oriented information can be incorporated in the deductive process

Several aspects of the partition model need further investigation. We mention here just two that we consider very important :

Queries and Updates : We have used closures to describe formally the query processing mechanism of the partition model. We think that the concept of closure is important to both, querying and updating, and that the two aspects cannot be studied separately [6].

Lossless Decompositions : Sometimes partitions are redundant, in the sense that there are more economical ways of denoting their blocks. For example, suppose that ab is a non-empty block of a composite partition AB. If AB=A then ab=a and, therefore, ab can be viewed as a redundant way of writing a. The basic question is how to decompose a given composite partition into a logically equivalent set of non-redundant partitions, using the external information [5].

We conclude this paper by a brief discussion of views and null values, two concepts that have received wide attention in the relational model. Given a data base on universe U, a view is a new data base on U derived from the given data base. Views are defined using relational operators, such as projection and join, and are intended to serve specific user needs. However, relational views present difficult problems, especially with respect to updating. These problems are due to lack of semantic information for correct update translation (see for example [1]). In the partition model, view updating is not seen as distinct from data base updating. In fact, the data base and the view, together, constitute just a larger data base. This is possible in the partition model because the data base schema is not fixed. Thus, view updating is data base updating [6].

The null-value problem is also due to the fact that the schema of a relational data base is fixed. Null values (of various kinds) are necessary, when we want to record data which do not "fit" in the data base schema. For example, when we want to record ac in a data base whose schema is {AB,BC}. This kind of problem does not exist in the partition model. Data, such as ac in our example, will be recorded under AC, the new schema will be {AB,BC,AC}, and the deduction mechanism of the prtition model will be put to work on the new data base, as usual.

BIBLIOGRAPHY

[1] Bancilhon F. and Spyratos N., "Update Semantics of Relational Views, ACM TODS 6:4 December 1981, pp. 557-575

[2] Maier D., "The Theory of Relational Databases", Computer Science Press 1983

[3] Reiter R., "On closed World Databases", In Gallaire and Minker, New York : Plenum Publ. Co. 1978, pp. 55-76

[4] Smith J.M. and Smith D.C.P., "Database Abstraction : Aggregation and Generalization", ACM TODS 2:2, June 1977, pp. 105-133.

[5] Spyratos N., "Lossless Decompositions and Normal Forms in the Partition Model", INRIA Research Report, to appear.

[6] Spyratos N., "Database Updating and Quering in the Partition Model", INRIA Research report, To appear

[7] Spyratos N., "A Lattice Theoretic Model of Data and Algebraic Logic", In preparation.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique